НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут телекомунікаційних систем
Кафедра Телекомунікаційних систем

«До захисту допущено»
Завідувач кафедри
_____ Л.О. Уривський
«___»_____20__ р.

Дипломна робота
на здобуття ступеня бакалавра
з напряму підготовки 6.050903 Телекомунікації
(172 Телекомунікації та радіотехніка)
на тему: «Аналіз застосування хмарних технологій для додатків Інтернету речей»

Виконав:
студент 4 курсу, групи ТС-51
Дикий Артем Ігорович _____
Керівник:
Доцент кафедри телекомунікаційних систем,
кандидат технічних наук, доцент
Мошинська А.В. _____
Рецензент:

_____ _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

Кафедра Телекомунікаційних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки– 6.050903 «Телекомунікації» (172 Телекомунікації та радіотехніка)

Програма професійного спрямування – «Телекомунікаційні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Л.О. Уривський

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Дикому Артему Ігоровичу

1. Тема роботи «Аналіз застосування хмарних технологій для додатків Інтернету речей», керівник роботи Мошинська, Аліна Валентинівна, Доцент кафедри телекомунікаційних систем, кандидат технічних наук, затверджені наказом по університету від «08» квітня 2019 р. № 1068-с

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Збір основних матеріалів для написання дипломної роботи	11.04.2019	Викон.
2	Визначення ролі хмарних обчислень в IoT	22.04.2019	Викон.
3	Розбір альтернативних методів побудови IoT мереж та додатків з використанням Cloud Computing	09.05.2019	Викон.
4	Аналіз додатків IoT; Перелік основних проблем IoT платформ та шляхи їх подолання	21.05.2019	Викон.
5	Формування вступної частини та підведення висновків	29.05.2019	Викон.
6	Компонування фінальної версії дипломної роботи та презентаційних плакатів	08.06.2019	Викон.

Студент А.І. Дикий

Керівник роботи А.В. Мошинська

РЕФЕРАТ

Текстова частина дипломної роботи: 58 с 11 рис., *2* табл., 12 джерел.

Мста роботи - с аналіз архітектури, моделі, та принципу взаємодії додатків Internet of Things з технологією Cloud computing.

Дана робота складається з чотирьох основних частин: в першій частині досліджено принцип роботи, а також аналіз архітектури та структури моделі додатків Інтернету речей, та їх складових елементів. Досліджено принцип роботи, а також розглянуто переваги та недоліки побудови додатків Інтернету речей.

В другій частині досліджено технологію Fog Computing, як подальший розвиток синергії технологій Internet of Things та Cloud computing.

В третій частині описуються переваги та недоліки побудови додатків Інтернету речей . Також розглянуто перспективні технології розвитку IoT.

В четвертій частині було проведено аналіз побудови складної архітектури технологій передачі даних IoT.

Ключові слова: IoT, Cloud, Fog Computing, Ege Computing.

ABSTRACT

Text part of the thesis: 58 of 11 rice, 2 tables, 12 sources.

Point of work - from analysis of architecture, model, and the principle of interaction of applications Internet of Things with technology Cloud computing.

This work consists of four main parts: the first part examines the principle of work, as well as analysis of the architecture and structure of the model Internet applications of things, and their constituent elements. The work principle is explored, and the advantages and disadvantages of constructing Internet applications of things are considered.

The second part examines the technology of Fog Computing, a further development of the synergy of Internet of Things and Cloud computing technologies.

The third part describes the advantages and disadvantages of building Internet applications. Also reviewed are promising technologies for the development of IoT.

In the fourth part, an analysis of the construction of a complex architecture of IOT data transfer technologies was conducted.

Key words: IOT, Cloud, Fog Computing, Ege Computing.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ADS	Autonomous Driving System – Автономні системи управління транспортом
API	Application Programming Interface– інтерфейс прикладного програмування
GDPR	General Data Protection Regulation— Загальний регламент про захист даних
IoT	Internet of Things— Інтернет речей
M2M	Machine-to-Machine — машино-машинна взаємодія
MCC	Mobile Cloud Computing— мобільні хмарні обчислення
MQTT	Message Queue Telemetry Transport— спрощений мережевий протокол
PaaS	Platform as a Service— модель надання хмарних обчислень
RFID	Radio frequency identification — радіочастотна ідентифікація.
SDK	Software Development Kit— набір із засобів розробки
SSL	Secure Sockets Layer— рівень захищених сокетів
STA	SmartThings Apps — додатки «розумних речей»
TLS	Transport Layer Security — захист на транспортному рівні
WAN	Wide Area Network — глобальна обчислювальна мережа
WLAN	Wireless Local Area Network — бездротова локальна мережа
WSN	Wireless sensor network — бездротова сенсорна мережа

ВСТУП

Розвиток ринку інформаційних технологій призвело до появи концепції Інтернету речей (Internet of things). Принцип IoT базується на взаємодії звичних для нас в побуті речей за допомогою високошвидкісних обчислювальних мереж. В широкому розумінні Інтернет речей - це не просто безліч різних приладів і датчиків, об'єднаних між собою дротяними і бездротовими каналами зв'язку і підключених до мережі Інтернет, а це більш тісна інтеграція реального та віртуального світів, в якому основну роль відіграє спілкування між людьми і всілякими пристроями..

Концепція Інтернету Речей з'явилася в 1982 році, коли модифікований автомат з газованою водою був підключений до Інтернету і був здатний повідомляти про наявність в ньому напоїв і їх температурі. У 1999 році Кевін Ештон запропонував термін «Інтернет Речей» для пов'язаних пристроїв. Базовою ідеєю IoT є надання можливості автономного обміну корисною інформацією між унікально ідентифікованими пристроями реального світу. Ці пристрої оснащені новітніми технологіями, такими, як радіочастотна ідентифікація (RFID) і бездротові мережі датчиків (WSNs), і в подальшому отримують можливість приймати самостійні рішення в залежності від того, який автоматизований процесс виконується.

На думку Роба Ван Краненбург IoT можна умовно розділити на 4 рівні.

- 1 рівень пов'язаний з ідентифікацією кожного об'єкта.
- 2 рівень надає з сервісом по обслуговуванню потреб споживача (можна розглядати як мережу власних «речей», приватний приклад - «розумний дім»).
- 3 рівень пов'язаний з урбанізацією міського життя. Тобто це концепція «розумного міста», де вся інформація, яка стосується жителів цього міста, стягується в конкретний житловий квартал, в Ваш будинок і сусідні будинки.
- 4 рівень - сенсорна планета.

Бажання багатьох користувачів відчути себе в ролі творців підштовхнуло деякі компанії до розробки спеціальних програмованих платформ. В результаті виявилося, що подібні розробки дозволили впоратися з різними завданнями, починаючи рішенням інфраструктурних концепцій і закінчуючи створенням інтерактивних об'єктів. У даній роботі я провів дослідження по визначенню особливостей IoT платформ з точки зору звичайного користувача і розробника, а також з’ясував особливості їх взаємодій з хмарними сервісами .

1 РОЛЬ ХМАРНИХ ОБЧИСЛЕНЬ В ІНТЕРНЕТІ РЕЧЕЙ

Інтернет речей (IoT) поступово трансформував спосіб виконання щоденних завдань. Окрім надання розумних рішень для житлових та житлових спільнот, IoT також використовується як інструмент у бізнес-середовищі різних галузей. Тим не менш, з великою кількістю даних, які генеруються IoT, багато навантаження припадає на інфраструктуру Інтернету. Це змусило підприємства та організації шукати варіант, який би зменшив це навантаження.

Намагаючись зрозуміти рішення IoT, важливо взяти до уваги один з основних компонентів - Cloud Backend.

Він несе відповідальність за прийом інформації з шлюзу IoT, зберігання і обробку їх у діючих ресурсах і надсилання їх до інтерфейсу користувача (веб-додаток/мобільний додаток/панель інструментів).

У деяких вдосконалених рішеннях IoT хмарні додатки IoT також підтримують такі можливості, як машинне навчання та штучний інтелект.

Такі нововведення в розробці додатків хмари IoT гарантують, що рішення IoT здатні вирішувати складні бізнес-проблеми в галузях промислової автоматизації, підключених автомобілів, пов'язаних медичних послуг і багато іншого.

Існує нерозривний зв'язок між IoT і Cloud. Дані, зібрані датчиками, досить великі у випадку промислового застосування IoT і шлюз не здатний обробляти і зберігати їх. Ці дані потрібно зберігати в захищеній базі даних і обробляти доступним і масштабованим способом. Саме тут з'являються хмарні додатки IoT .

Хмара підключається до шлюзу IoT через Інтернет і отримує всі дані, що надходять до шлюзу датчиками. Існує декілька протоколів, які з'єднують шлюзи з облачними програмами IoT, і найбільш поширеним серед них є MQTT.

Датчики збирають і подають дані у всі часи і цей величезний шматок даних після агрегації і деяка попередня обробка передається в хмару для зберігання і обробки.

1.1 Взаємозв’язок IoT і Cloud Computing

Хмарні обчислення, а також IoT, працюють у напрямку підвищення ефективності повсякденних завдань, і обидва мають взаємодоповнюючі відносини. З одного боку, IoT генерує багато даних, а з іншого боку, хмарні обчислення прокладають шлях для цих даних. Є багато провайдерів хмар, які користуються цим, щоб забезпечити модель, що використовує оплату за використання, де клієнти сплачують за використані конкретні ресурси. Крім того, хмарний хостинг як послуга додає цінності стартапу IoT, забезпечуючи економію від масштабу, щоб зменшити їх загальну структуру витрат.

На додаток до цього, хмарні обчислення також дають змогу краще співпрацювати для розробників, що є порядком дня у просторі IoT. Допомагаючи розробникам зберігати та віддалено доступ до даних, хмара дозволяє розробникам реалізовувати проекти без затримки. Крім того, зберігаючи дані в хмарі, компанії IoT можуть отримати доступ до великих масивів даних. Щоб, наочно встановити зв'язок між IoT і хмарою, нижче наведена таблиця, яка відображає, що вони чудово доповнюють одне одного.

Таблиця 1.1 Основні особливості технологій Internet of Things та Cloud Computing

Параметр	Інтернет речей	Хмарні обчислення
Масиви даних	Виступає джерелом масивів даних	Виступає як спосіб або засіб для керування масивами даними
Досяжність даних	Дуже обмежена	Велика дальність поширення

Продовження Таблиці 1.1

Параметр	Інтернет речей	Хмарні обчислення
Сховище даних	Обмежене або майже відсутнє	Велике, практично не закінчується
Роль Інтернету	Виступає як точки конвергенції	Виступає як засіб надання послуг
Обчислювальні можливості	Обмежені	Практично необмежені
Компоненти	Запускається на апаратних компонентах	Запускається на віртуальних машинах, які імітують апаратні компоненти

Так само, як хмарні обчислення будуються на принципах швидкості та масштабу, додатки IoT будуються на принципі мобільності та широкомасштабних мереж. Отже, дуже важливо, щоб як хмара, так і IoT формували хмарні додатки IoT, щоб максимально використати їх комбінацію. На додаток до цього, ще кілька показників, чому хмара важлива з точки зору успіху IoT.

Хмара як технологія дає можливість IoT вийти за межі звичайної техніки, наприклад, кондиціонерів, холодильників тощо. Зі збільшенням мініатюризації та переходу з 4G на більш високі швидкості Інтернету, хмара дозволить розробникам швидко розвантажувати великі за обсягом обчислювальні процеси.

Нині багато нововведень у сфері IoT розглядають хостинг-послуги plug-and-play. Саме тому хмара ідеально підходить для IoT. Хостинг-провайдери не повинні залежати від масового обладнання або навіть від будь-яких апаратних засобів, які не підтримують вимоги до спритних пристроїв IoT. Хмара діє як міст у вигляді посередника або комунікатора, коли мова йде про IoT. Багато потужних API, такі як Cloudflare, CloudCache і Dropstr, що використовують хмарні комунікацій, що дозволяє легко зв’язуватись із смартфоном.

Було б справедливо сказати, що хмара може прискорити зростання IoT. Однак розгортання хмарних технологій також має певні проблеми та недоліки. Не тому, що хмара є недосконалою як технологія, проте комбінація хмари та IoT може обтяжувати користувачів деякими перешкодами.

Залежно від характеру реалізації IoT, хмара може мати різну ступінь складності. У простих додатках хмара може складатися з бази даних, в якій зберігаються дані, зібрані IoT, а також інформація користувачів, які мають право доступу/зміни даних.

У великих і складніших реалізаціях хмарні додатки IoT можуть також мати можливість машинного навчання, виконання аналітики, генерації звітів і багато іншого.

Деякі протоколи, такі як MQTT, Websocket, CoAP і AMQP, використовуються для розробки потужного та безпечного інтерфейсу, який полегшує безперешкодний зв'язок між датчиками та хмарою. Для того, щоб гарантувати відсутність втрати даних під час важкого припливу даних, розроблено надійну базу даних.

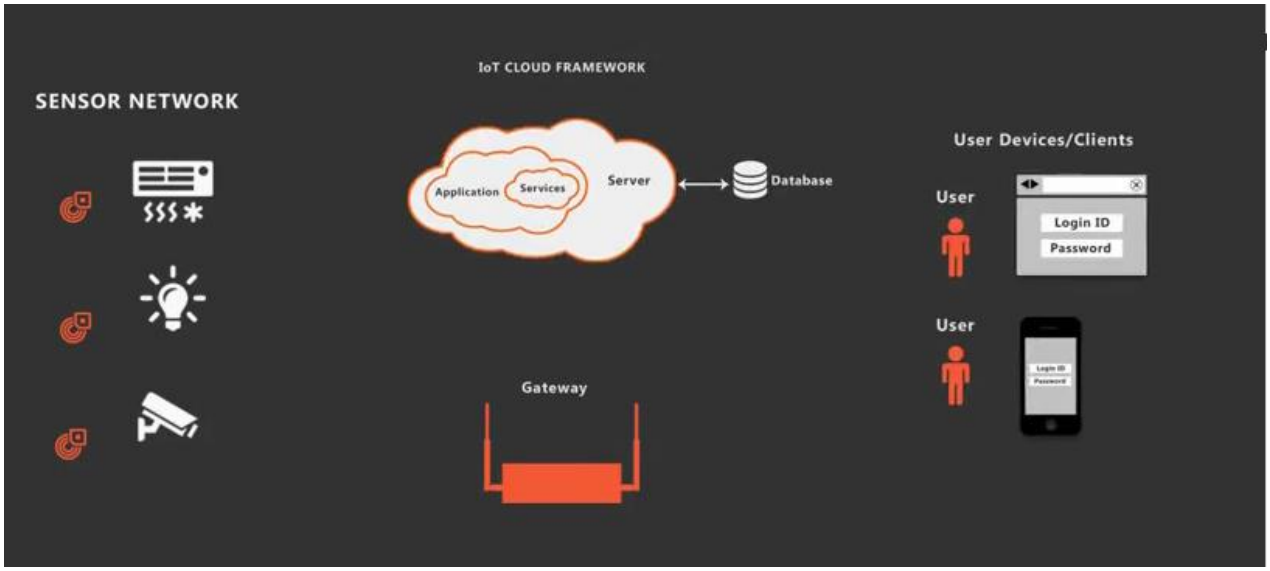

Рисунок 1.1 Іллюстрація базових складових IoT

- Дизайн баз даних. IoT включає в себе багато даних, які необхідно зберігати; отже, база даних дуже важлива для будь-якої реалізації IoT.

Найкращі практики включають аналіз кількості сенсорних вузлів і відповідну розробку бази даних. Також необхідно оптимізувати управління даними.

- Масштабованість сервера. Залежно від поточної та прогнозованої кількості кінцевих користувачів програми IoT, сервер повинен мати положення для автоматичного масштабування.
- Клонування додатків. Клонування додатків - це функція, яка допомагає уникнути перевантаження системи у разі збільшення трафіку. Це ефективний спосіб обробки важкого трафіку і повинен бути включений в хмарі програми IoT.
- Захист програм. По-перше, пакети даних, що передаються по мережах, повинні бути зашифровані, а по-друге, TLS/SSL-сертифікат повинен бути включений, щоб можна було запобігти віддаленому доступу датчиків і пристроїв IoT.

Кінцеві користувачі IoT можуть бути де завгодно в світі, і їх потрібно обслуговувати цілодобово, незалежно від їх географічного розташування та часових поясів.

IoT і хмара прекрасно доповнюють один одного, оскільки перший збирає дані і служить джерелом даних, хмара виступає як призначення, так і дистриб'ютор.

Деякі переваги хмари в екосистемі IoT

- Забезпечує зберігання та обробку даних IoT. IoT має величезний потенціал, і в найближчому майбутньому ми могли б бачити всі види фізичних осіб, пов'язаних один з одним. Це вимагатиме великих обчислювальних потужностей, і це може забезпечити лише хмара.
- Розширена аналітика та моніторинг. З безліччю «речей», які зараз підключені, існує потреба у постійному аналізі та моніторингу, щоб забезпечити безперешкодний досвід користувачів Інтернету. Розширена розробка хмарних додатків гарантує, що хмара обладнана такими можливостями.
- Більш рівний зв'язок між пристроями. У IoT, датчики взаємодіють не тільки з користувачами, вони також взаємодіють один з

одним. Застосування IoT Cloud разом з шлюзом IoT гарантує, що різні датчики та виконавці можуть взаємодіяти один з одним без будь-якої несумісності.

Розробка обласних додатків IoT підтримується деякими дійсно просунутими та перевіреними технологіями, такими як Amazon EC2 у поєднанні з EBS (Elastic Block Store). Кілька інших - G Suite від Google і Microsoft Azure.

З приходом до революції багатьох компаній з розробки обласних програм IoT , ми можемо очікувати, що багато цікавих нововведень через кілька років, і хмара буде в їхній основі.

1.2 Проблеми при використанні хмари та IoT

Обробка великої кількості даних може бути надзвичайно складною, особливо якщо на одночасно в системі мільйони пристроїв. Це пояснюється тим, що на карту поставлено загальну продуктивність програм. Отже, після NoSQL рух даних може бути хоча й корисним, проте не випробуваний в довгостроковій перспективі. Ось чому не існує жодного ефективного методу для керування великими даними хмари.

Мережеві та комунікаційні протоколи. Хмара та IoT включають зв'язок між машинами та різними типами пристроїв, що мають різні протоколи. Управління цим варіантом може бути складним, оскільки більшість прикладних областей не передбачає мобільності. На сьогоднішній день WiFi і Bluetooth використовуються як надійні віхи для полегшення мобільності до певної міри.

Сенсорні мережі посилили переваги IoT. Ці мережі дозволили користувачам вимірювати, виводити і розуміти делікатні показники з навколишнього середовища. Тим не менш, своєчасна обробка великої кількості даних датчиків була головною проблемою. Хоча хмара надає нову можливість у зведенні даних датчиків, вона також перешкоджає прогресу через проблеми безпеки та конфіденційності.

1.3 Типи моделей хмарних обчислень для рішень IoT

Існують три типи моделей хмарних обчислень для різних типів підключеного середовища, які зазвичай пропонуються постачальниками хмарних послуг:

1. Інфраструктура як послуга:
    a. Надає віртуальні сервери і сховища підприємствам. В основному, дає доступ до мережевих компонентів, таких як комп'ютери, сховище даних, мережні підключення, балансування навантаження і пропускну здатність.
    b. Збільшення критичних даних всередині організації призводить до вразливостей безпеки, а IaaS може сприяти розповсюдженню критичних даних у різних місцях віртуально (або фізично) для поліпшення безпеки.
2. Платформа як послуга
    a. Це дозволяє компаніям створювати програмне забезпечення та програми з інструментів та бібліотек, що надаються постачальниками хмарних сервісів.
    b. Це усуває основні потреби управління апаратними засобами та операційними системами і дозволяє підприємствам більше зосереджуватися на розгортанні та управлінні програмним забезпеченням або додатками.
    c. Це зменшує турботу про підтримку операційної системи, планування потужності та інших важких навантажень, необхідних для запуску програми.
3. Програмне забезпечення як послуга
    a. Забезпечує повне програмне забезпечення або програму, яку запускає та підтримує лише постачальник хмарних послуг.

1.4 Горизонтальна структура побудови IoT додатків

Горизонтальна, забезпечує спільні функції служб, які дозволяють застосуванню в декількох доменах, використовуючи загальну структуру і єдині API.

Використання цих стандартизованих API дозволяє набагато простіше для постачальників рішень M2M / IoT вирішувати складні та різнорідні варіанти підключення, абстрагуючи деталі використання базових мережевих технологій, базових транспортних протоколів та серіалізації даних. Все це обробляється службовим шаром OneM2M без необхідності, щоб програміст став експертом у кожному з цих шарів. Таким чином, розробник програми може зосередитися на процесі / бізнес-логіці випадку використання, щоб бути реалізований і не потрібно турбуватися про те, як саме основні шари працюють. Це дуже схоже на написання файлу до файлової системи, не турбуючись про те, як працюють жорсткі диски та їх інтерфейси.

Таким чином, службовий шар IoT, зазначений в одному M2M, можна розуміти як розподілену операційну систему для IoT, що забезпечує однорідні інтерфейси API для програм IoT подібно до того, як це робить мобільна ОС для системи екологічного смартфона.

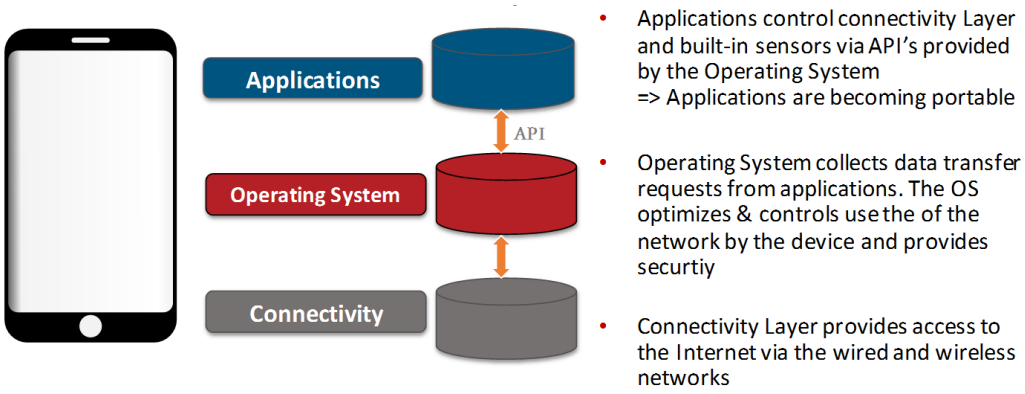

Рисунок 1.3 Приклад горизонтальної архітектури додатків IoT

Наприклад, такі "вертикальні" домени є ізольованими елементами, що ускладнює обмін даними між собою. Використання «горизонтальної»

архітектури дозволяє забезпечити безперервну взаємодію між додатками та пристроями. У наведеному нижче випадку використання, програма безпеки виявляє, що, коли ніхто не знаходиться в будівлі, він запускає вимикання світла і зупиняє кондиціонер.

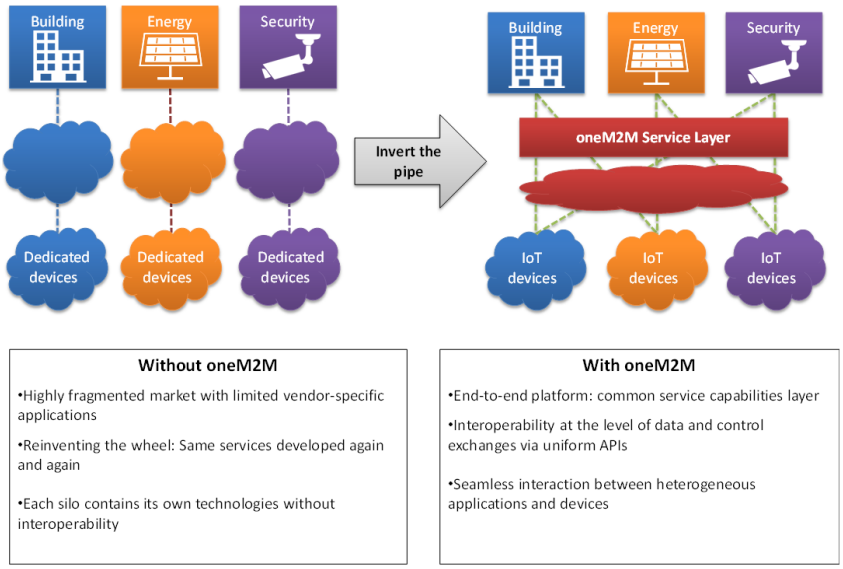

Рисунок 1.4 Приклад горизонтальної архітектури додатків IoT з використанням технології oneM2M

1.5 IoT-платформи і додатки з розвиненою аналітикою і моделюванням

IoT-платформи – об'єднує власне "речі" і "інтернет". По суті - це ключовий інструмент розробки IoT-додатків і сервісів, що поєднує фізичні об'єкти і Мережу.

На думку авторів "IoT Analytics", повноцінною IoT-платформою слід вважати таку платформу, яка дозволяє розробляти відповідні додатки чи рішення (IoT Application Enablement Platform).

Для наочності нижче приведено чотири типи платформ, які називають "IoT-платформами", проте вони не цілком підходять під класифікацію IoT Analytics:

- Consumer/Enterprise software extensions. Існуючі пакети корпоративного програмного забезпечення і операційні системи

типу MS Windows 10 стають все більш відкритими для інтеграції IoT-пристроїв. В даний час ця область ще недостатньо розвинена, щоб називатися IoT-платформою, але майбутнє у неї вельми перспективне.

- IaaS backends. Інфраструктура-як-сервіс-сервери, що надають хостинг-простір і обчислювальні потужності для додатків і сервісів, раніше оптимізували для десктопів і мобільних додатків, але зараз в фокус потрапив і IoT (приклад - IBM Bluemix).
- Hardware-specific software platforms. Деякі компанії, що продають розумні гаджети, створюють власний програмний бекенд і подають його, як IoT-платформу. Але так як ця платформа носить закритий для всіх інших характер, правомірність такого найменування сумнівна (наприклад - Google Nest).
- Connectivity/M2M platforms. Платформи в своїй роботі фокусуються на зв'язку розумних об'єктів через телекомунікаційні мережі, але рідко на обробці сигналів від датчиків (приклад такої платформи: Sierra Wireless з продуктом AirVantage).

Отож для розуміння природи та суті IoT-платформ виділили вісім компонентів повноцінної IoT-платформи:

1. Зв'язок і нормалізація (Connectivity & normalization): зведення різних протоколів і форматів даних в один "програмний" інтерфейс, гарантуючи точну передачу даних і взаємодію з усіма пристроями.
2. Управління пристроями (Device management): забезпечення належного функціонування підключених "інтернет-речей", їх конфігурація, безперебійність роботи, додавання патчів і оновлень. Причому, не тільки ПО власне "речей", але і додатків, що працюють на пристрої або прикордонних шлюзах.
3. База даних (Database): тут все досить зрозуміло і прозоро – масштабоване сховище даних від "речей". Вимоги до цих даних,

впорядкування при обробці і перенесенні даних з, наприклад, різних "платформ".

4. Обробка та управління діями (Processing & action management): дані, отримані від "речей" в кінцевому підсумку впливають на події в реальності. З цього "платформа" повинна вміти будувати процеси, "тригери подій" та інші "розумні дії" на основі конкретних даних датчиків.
5. Аналітика (Analytics): дані від "речей" є цінними самі по собі. Тому існування комплексу засобів їх аналізу є обов'язково вимогою до "платформи". Якщо сюди включити ще й кошти кластеризації даних і глибокого машинного навчання аж до прогнозуючої аналітики, то цінність "платформи" очевидно зростає.
6. Візуалізація (Visualization): всю перераховану вище аналітику і решту даних потрібно показати таким чином, щоб людям було зрозуміліші дані процеси. Це побудова графіків, моделей, чи ж візуалізація того, що відбувається з "речами". Або ж просто зручний інтерфейс.
7. Додаткові інструменти (Additional tools): набір інструментів, який дозволяє розробникам IoT створювати прототипи, тестувати і пробувати різні системи.
8. Зовнішні інтерфейси (External interfaces): інтеграція за допомогою платформи - одна з головних можливостей. Світ інтернет-розробки нині не терпить замкнутих рішень. Завжди може знадобитися передача і обмін зі сторонніми системами. Тому справжня IoT-платформа повинна мати інтерфейси прикладного програмування (API), комплекти розробки програмного забезпечення (SDK) і шлюзи.

1.6 Висновки з розділу 1

Введення хмарних обчислень – дозволяє постачати обчислювальні потужності на вимогу, сховища баз даних, програм та IT-ресурсів. Вона дозволяє організаціям споживати обчислювальний ресурс, як віртуальну машину (ВМ) замість створення обчислювальної інфраструктури на місці.

Багато технологій, таких як Amazon, Alibaba, Google і Oracle, створюють засоби машинного навчання за допомогою хмарних технологій, щоб запропонувати широкий спектр рішень для підприємств у всьому світі.

2 «ТУМАННІ» ОБЧИСЛЕННЯ ТА ІНТЕРНЕТ РЕЧЕЙ

Туманні обчислення (Fog Computing)- термін, створений Cisco у 2012 році, також знаний як "фоггінг", по суті означає розширення обчислень до краю мережі підприємства, замість розміщення та роботи з централізованої хмари. Це полегшує локальну обробку даних в інтелектуальних пристроях і згладжує роботу обчислювальних, сховищ і мережевих послуг між кінцевими пристроями і обчислювальними центрами хмарних обчислень.

Туманні обчислення підтримують IoT, 5G, штучний інтелект (AI) та інші програми, які вимагають наднизьких затримок, високу пропускну здатність мережі, обмеження ресурсів та додаткову безпеку.

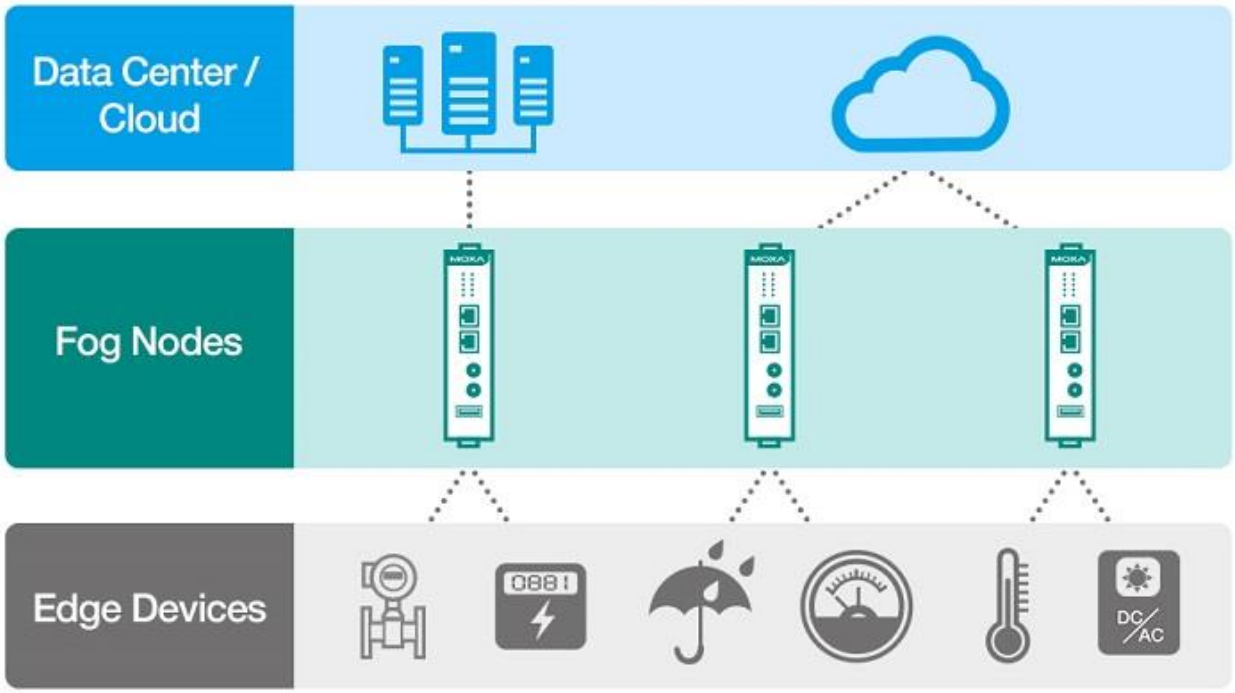

Рисунок 2.1 Структурна схема Fog Computing

Як Cloud, так і Fog Computing, використовують подібні IT-ресурси: обчислювальні пристрої (сервери і процесори комп'ютерів користувачів), вузли комутації мережі і системи зберігання даних. Однак, розширення хмари до меж мережі не зводиться лише до масштабування цієї хмари.

2.1 Різниця туманними і межовими обчислення

Периферійні (межові) обчислення - це підхід, пов'язаний з обробкою даних на межі мережі, де створюються дані, а не в централізованому сховищі, призначеному для обробки даних.

Межові обчислювальні системи являють собою розподілену відкриту IT-архітектуру, в якій застосовується децентралізована обробка і забезпечується підтримка технологій мобільних обчислень Інтернету речей. При використанні периферійних обчислень дані обробляються самим пристроєм, локальним комп'ютером чи сервером, а не передаються в центр обробки даних.

Межові обчислювальні системи забезпечують прискорення потоків даних, включаючи обробку даних в реальному часі без затримки. Вони дозволяють інтелектуальним програмам і пристроям реагувати на дані практично відразу ж після їх створення, виключаючи будь-які затримки. Це критично важливо для розвитку таких технологій, як автомобілі з автопілотом, а також забезпечує важливі переваги для організацій.

Межові обчислення забезпечують ефективну обробку великих обсягів даних поруч з джерелом, знижуючи завантаження інтернет-каналів. З одного боку, це дозволяє скоротити витрати, а з іншого - ефективно використовувати програми віддалено. Крім того, здатність обробляти дані без їх приміщення в загальнодоступне хмара забезпечує додатковий рівень захисту конфіденційних даних.

В свою чергу туманні обчислення завжди використовують межові обчислення, але не навпаки. Фоггінг - це архітектура системного рівня, що надає інструменти для розповсюдження, організації, керування та забезпечення ресурсів і послуг у мережах і між пристроями, які знаходяться на периферії.

Архітектури межових обчислень розміщують сервери, програми або невеликі хмари на периферії. Туманне обчислення має ієрархічну і плоску

архітектуру з декількома шарами, що утворюють мережу, в той час як межове обчислення покладається на окремі вузли, які не утворюють мережу.

Туманне обчислення має широкі можливості однорангового взаємозв'язку між вузлами, де кожна межа виконує свої вузли в силосах (фрагментах мережі ізольованих одне від одного), що вимагає перенесення даних через хмару для однорангового трафіку. Також варто відмітити що туманні обчислення включають використання хмарних сервісів, тоді як межові обчислення виключають їж використання взагалі.

2.2 Основні архітектурні відмінності Fog від Cloud

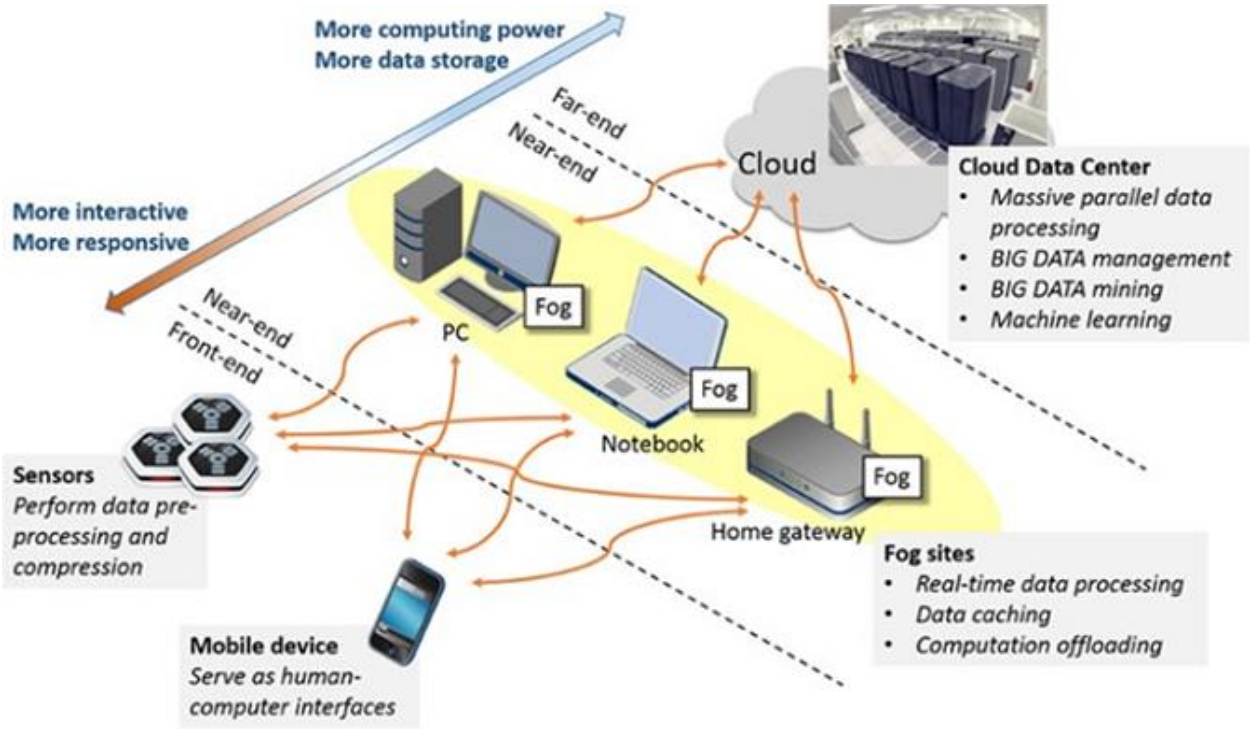

Рисунок 2.2 Архітектура мережі Fog Computing[1]

Основна відмінність туманних обчислень від хмарних обчислень полягає в тому, що хмара є централізованою системою, тоді як туман, по суті, є розподіленою децентралізованою інфраструктурою.

Вузли туману:

- Отримувати відклик з пристроїв IoT за допомогою будь-якого протоколу в режимі реального часу.

- Запускати IoT -додатки для контролю та аналізу в реальному часі, при чому час відклику сягає мілісекунд.
- Забезпечують тимчасове зберігання даних, зазвичай 1-2 години.
- Надсилати у хмару періодичні зведення даних.

Хмарна платформа:

- Отримує та агрегує зведення даних з багатьох вузлів туману.
- Виконує аналіз IoT -даних та даних з інших джерел для отримання ділової інформації.
- Може надсилати нові правила застосування до вузлів туману на основі цих даних.

Основні архітектурні відмінності Fog від Cloud полягають у:

- Забезпечення якості послуг (QoS, Quality of Service), що вимагає динамічної адаптації додатків до стану мережі.
- Відстеження місця розташування (Location Awareness) для того, щоб підтримувати стабільність роботи програми в умовах мобільності терміналу.
- Відстеження контекстної інформації (Context Awareness), тобто здатність виявляти наявність доступних ресурсів поблизу, щоб задіяти їх в роботі додатка, з можливістю горизонтального взаємодії.

В архітектурі Fog мережеві вузли (Fog Sites), розташовані ближче до хмарних дата-центрів, та мають більшу обчислювальну потужність з більшим обсягом даних в системах зберігання. Мережеві вузли, розташовані ближче до сенсорів інтернету речей і мобільних пристроїв, мають більшу інтерактивність і швидший відгук. Відмінною особливістю Fog є те, що в якості мережевого вузла можуть виступати пристрої користувача, такі як персональні комп'ютери, домашні шлюзи, телеприставки і мобільні пристрої. Щоб пристрій користувача міг працювати як вузол мережі Fog, користувач повинен дати оператору зв'язку відповідний дозвіл на використання

обчислювальної потужності свого гаджета в фоновому режимі, в обмін на різні пільги з боку оператора.

Розробники портують або записують програми IoT для вузлів на межі мережі. Туманні вузли, найближчі до краю мережі, приймають дані з пристроїв IoT. Потім - і це є надзвичайно важливим - туманні IoT додатки направляють різні типи даних до оптимального місця їх аналізу.

- У більшості випадків чутливі до часу дані аналізуються на вузлі туману, найближчому до джерел генерації даних.
- Централізований кластер(вузол) агрегації використовується для аналізу даних, які можуть чекати секунди або хвилини, після чого виконується дія.
- Для аналізу та зберігання даних у хмару надсилаються дані найменш чутливі до часу. Прикладом тут може бути кожен з туманних вузлів, що відправляють періодичні зведення даних до хмари для їх аналізу.

2.3 Приклади застосування Fog Computing

Автономні системи управління транспортом (ADS, Autonomous Driving System). ADS використовують різні багаторежимні сенсори, технології комп'ютерного зору і аналізу зображень, супутникове та мережеве позиціонування на картах і Інтелектуальне аналітику, на базі яких ADS допомагає керувати водію або управляє самостійно рухається транспортним засобом. У таких додатках потрібна висока швидкодія, тому Fog-вузол з елементами штучного інтелекту необхідно розміщувати безпосередньо в транспортному засобі.

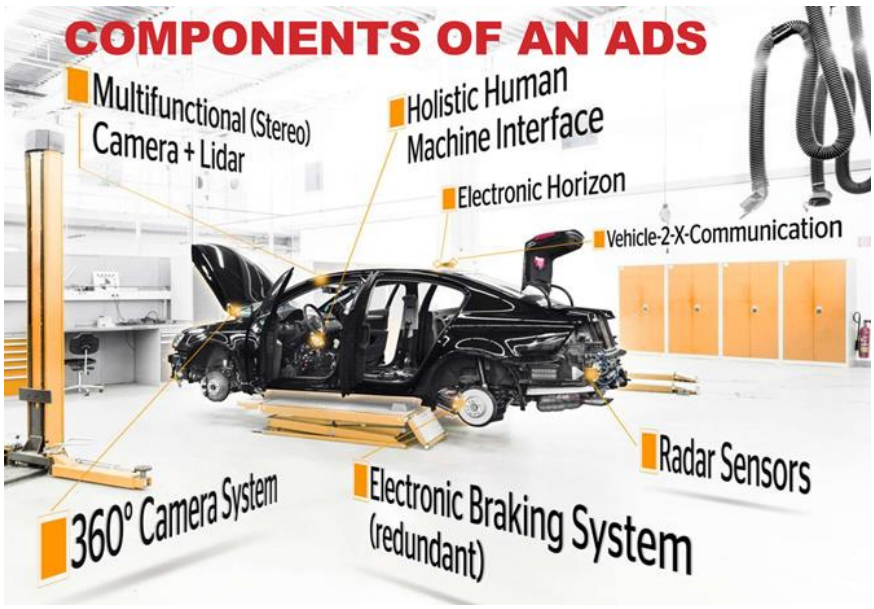

Рисунок 2.3 Компоненти Fog-вузлів в ADS

Fog-системи в електроній охороні здоров'я (eHealth). Fog-системи в медицині застосовуються в тих випадках, коли необхідно провести оперативний аналіз отриманих даних з датчиків носимих пацієнтом і вжити негайних дій відповідно до плану лікування .

Наприклад подекуди, Fog-технології вже застосовуються для контролю стану хворих на діабет і автоматичного введення ін'єкцій. Сенсор на тілі пацієнта визначає критичне значення вмісту цукру в крові, і через Fog-мережу видає сигнал на виконання ін'єкції за допомогою мікрошприца, також розташованого на тілі пацієнта. Таким чином, пацієнт позбавляється від необхідності постійно робити виміри і ін'єкції самому.

Fog-проекти хмарних провайдерів. У 2016 році три найбільших провайдера хмарних платформ - Amazon, Google і Microsoft - почали кілька проектів використання Fog Computing в своїх екосистемах IoT, в яких застосовується т.зв. «Безсерверна архітектура» (serverless architecture).

Безсерверна архітектура дозволяє виконувати вихідний код тисяч і мільйонів користувачів (зокрема, fog-пристроїв) всередині обчислювального середовища, не піклуючись про масштабування ресурсів.

- Компанія Microsoft анонсувала підтримку функцій Azure (Azure Functions) всередині платформи розробки SDK (Software Development Kit).

Функції Azure спочатку були введені в сімейства хмарних продуктів з безсерверної архітектурою (Serverless Architecture), розроблених в Microsoft.

- Компанія Amazon розробила платформу Greengrass з підтримкою т.зв. Lambda-функцій (безсерверної архітектури) в пристроях IoT при взаємодії з хмарної платформою AWS. Greengrass - це контейнер виконання програмного модуля, який може бути запущений безпосередньо на Fog-пристрої, а не на сервері в дата-центрі. Пристрої з Greengrass можуть обмінюватися інформацією між собою незалежно від наявності зовнішнього інтернету, тобто горизонтально між Fog-пристроями за допомогою різних радіо-протоколів інтернету речей.
- Google представив платформу для інтернету речей Android Things з підтримкою мікрокомп'ютерів Intel Edison і Joule 570x, NXP Pico i.MX6UL і Argon i.MX6UL, а також Raspberry Pi 3. Fog-додатки розробляються на платформі Android Studio для будь-якого з цих пристроїв. Android Things також забезпечує інтеграцію з Google Play і всієї екосистемою Android, на якій зараз працюють 90% смартфонів у світі. Таким чином, система Android Things дає можливість будь-якому Android-смартфону або планшету працювати в якості Fog-вузла.

2.4 Висновки з розділу 2

Розвиток IoT показав необхідність фільтрації і попередньої обробки даних перед відправкою в хмару. В основному, це такі програми:

- Додатки, що вимагають низької і передбачуваної затримки передачі інформації по мережі, наприклад, ігрові додатки або відеоконференції.
- Додатки для транспорту, такі як: безпілотні автомобілі, швидкісні потяги, інтелектуальні транспортні системи і ін.
- Додатки, що вимагають локальної обробки даних в реальному часі, такі як: інтелектуальні системи електропостачання (Smart Grid), інтелектуальні транспортні системи (ITC), геофізична розвідка надр,

управління трубопроводами, сенсорні мережі моніторингу навколишнього середовища та ін.

Fog не є альтернативою для Cloud. Навпаки, Fog плідно взаємодіє з Cloud, особливо в адмініструванні і аналітиці даних, і така взаємодія породжує новий клас додатків.

3 ПРОГРАМНИЙ АНАЛІЗ ВИКОРИСТАННЯ IoT-ДОДАТКІВ ДЛЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ І КОНФІДЕНЦІЙНОСТІ

Крім того, новий Європейський регламент загального захисту даних (GDPR) накладає обмежувальні вимоги на обробку даних, особливо коли надходять дані по моніторингу людини. Дотримання вимог GDPR є особливо складним у середовищі IoT, оскільки важко отримати згоду, необхідну для обробки персональних даних. Тому, в процесі проектування системи IoT необхідна стадія поінформованості GDPR, щоб проаналізувати її відповідність з урахуванням використовуваних технологій, та обробки масивів даних. Проблеми конфіденційності користувачів виникають навіть при створенні інформації або висновків про поведінку користувача.

Однак безпеку та конфіденційність можна також покращити за допомогою розподілених систем IoT. GDPR забороняє передачу даних країнам, які не мають достатнього рівня захисту даних. Наразі провайдери хмари можуть розгортати свою інфраструктуру центрів обробки даних із "безпечних" регіонів. Таким чином, спільна робота між об'єктами допомагає захистити конфіденційні дані, зберігаючи їх у джерелі, аніж передаючи цю інформацію в хмару.

IoT поставив питання про безпеку та конфіденційність цифрового простору. Програма аналізу має вирішальне значення у визначенні цих питань, проте застосування та обсяг аналізу програм в IoT залишається в значній мірі невивченим технічним співтовариством. Тому доречно розглянути питання конфіденційності та безпеки в IoT, які вимагають програмно-аналітичних методів з акцентом на виявлені атак проти цих систем і оборони, реалізовані нині.

Впровадження пристроїв IoT в громадські та приватні простори змінюють наш спосіб життя. Наприклад, домашні програми, які об'єднують смарт-замки, термостати, перемикачі, системи спостереження та прилади, дозволяють користувачам контролювати та взаємодіяти зі своїм житловим

простором з будь-якого місця. Незважаючи на те, що промисловість і користувачі сприйняли IoT, були висловлені занепокоєння щодо безпеки та конфіденційності цифрових розширених просторів. У середовищах IoT обов'язково є доступ до функцій, які, якщо ними зловживають, ставлять під загрозу безпеку користувачів, наприклад, розблокують двері, коли користувач не знаходиться вдома або створює небезпечні умови, вимикаючи тепло в холодну погоду. Крім того, ці мережеві системи мають доступ до приватних даних, які, при їх витоку, спричинять питання конфіденційності, наприклад, інформацію про те, коли користувач спить або хто і коли інші перебувають вдома.

Критика IoT полягає в тому, що існуючі платформи втрачають необхідні інструменти і послуги для аналізу безпеки і конфіденційності. Такі зауваження не залишилися непоміченими. Роботи в цій області використовують методи програмного аналізу для розробки і побудови алгоритмів, які ідентифікують уразливості і небезпечну поведінку в рамках цільової IoT програми платформи.

Платформам програмування IoT присутні унікальні характеристики і проблеми в аналізі програм у порівнянні з іншими платформами. По-перше, у випадку з Android, є чітко визначене проміжне представлення (ПП), а аналіз може безпосередньо проаналізувати ПП-код. Проте платформи IoT програмування різноманітні, і кожна використовує свою, власну мову програмування. По-друге, IoT інтегрує фізичні процеси з цифровим підключенням через різноманітний набір пристроїв, кожен з яких має інший набір внутрішніх станів пристроїв (наприклад, заблокований/розблокований); таким чином, виявлення проблем безпеки та конфіденційності через ці фізичні стани є досить тонким. Наприклад, злодій може проникнути в будинок, змінивши значення температури термостата, що призводить до відкриття вікон, коли температура досягає порогового значення. Нарешті, кожна платформа програмування IoT має свої особливості, які можуть стати проблемою для програмного аналізу.

Кожна з цих особливостей робить програмний аналіз більш складним і вимагає спеціальної обробки. Завдяки цим особливостям, забезпечення безпеки, охорони та конфіденційності систем IoT не є тривіальним завданням.

3.1 Проектування захищених додатків IoT

На етапі планування або розробки IoT додатків повинна існувати офіційна оцінка безпеки і конфіденційності програми. Розробка програм IoT вимагає підходу "Безпечний Дизайн". Цей підхід передбачає врахування вимог безпеки всіх функцій IoT додатків як частини етапу проектування, а не прийняття та застосування функцій безпеки пізніше, в процесі розробки. Це дорожче і займає більше часу для виправлення проблем безпеки на більш пізніх етапах розробки, як і будь-яка інша помилка або проблема. Тому на етапі проектування додатків необхідно розглянути та спланувати всі можливі вимоги безпеки для програми IoT.

На етапі проектування варто переглянути вимоги безпеки для IoT додатків, виконавши перевірку вимог безпеки:

Необхідно створити безпечний механізм скидання пароля. Цей процес зазвичай не надто зручний для користувачів, але простий, і як наслідок ненадійний процес скидання пароля може забезпечити легкий бекдор в систему.

Необхідно чітко спланувати функції керування обліковими записами користувачів. Тобто переконатися, що програма IoT матиме відповідний рівень налаштування облікового запису.

Створити структуру облікового запису користувача, щоб обмежити привілеї адміністративного облікового запису на основі потреб. Доцільно відокремити адміністративні дії та права від стандартних облікових записів користувачів, оскільки ця конфігурація обмежує ризик неправильної конфігурації користувачами, що може призвести до серйозних порушень безпеки.

Визначити, як будуть зберігатися паролі облікових записів у програмі. Слід уникати зберігання звичайних текстових паролів у базах даних.

Також існує можливість додавання функції двохфакторної автентифікації, особливо для додатків, які оброблятимуть конфіденційні дані, та мають бути доступні з ненадійних мереж.

3.2 Програмний аналіз додатків IoT’

Методи аналізу додатків використовують вихідний код IoT додатків для досягнення різноманітних цілей, таких як безпека додатків.

Багато з цих цілей залишаються відкритими проблемами; Таким чином, розуміння цілей може керуватися майбутньою роботою:

Витоки чутливих даних. Пристрої IoT мають доступ до даних, які можуть бути виключно приватними, наприклад, двері заблоковані або розблоковані, а користувачі присутні вдома або ні. Платформи IoT забезпечують лише поверхневий контроль доступу до конфіденційної інформації та забезпечують обмежений контроль над тим, як ця інформація використовується. Наприклад, якщо користувач надає доступ до лічильника енергії, користувач не може дізнатися, чи буде програма надсилати споживання енергії розробникові додатків, рекламодавцям або іншим особам.

Засоби запобігання зловживанням. IoT додатки обов'язково мають доступ до функцій, які, якщо ними зловживають, ставлять під загрозу безпеку користувача, наприклад, розблокують двері, коли користувач не перебуває вдома, або створює небезпечні або шкідливі умови, якщо ввімкніть розумну піч . Тому дуже важливо запобігти зловживанню програмними додатками IoT, забезпечуючи, щоб ці програми працювали з пристроями відповідно до правил безпеки та функціональних властивостей.

Неправильне використання допусків. Модель допусків платформи IoT визначає доступ додатків до таких подій як зміна стану пристрою. Проте IoT додатки можуть зловживати моделями допусків. Це може відбутися з двох

основних причин. По-перше, модель допусків може бути поверхневою і конфліктною з допусками пристроїв; наприклад, додаток, що надає допуск на використання дверного замка, надає доступ як до дверного замка, так і до дій щодо розблокування дверей в цілому. По-друге, додаток може змусити користувачів встановити непотрібні та небезпечні допуски пристроїв; наприклад, додаток димової сигналізації може запитувати дозвіл на вимкнення у камери безпеки, навіть якщо додаток не потребує дозволу на це.

Походження даних. Так як IoT додатки виконують все більш різноманітні дії, атаки та несправності вимагають розслідування. Щоб вирішити цю проблему, системи походження використовують програмні засоби, які збирають інформацію про IoT додатки, щоб побудувати повну і точну карту поведінки. Після цього вони об'єднують цю інформацію в структуру даних, таку як графіки походження для криміналістичної та системної діагностики. Наприклад, система походження, призначена для додатків IoT, може забезпечити повну історію дій пристрою та подій, які можуть бути використані для виявлення причини атаки.

3.3 Проблеми в аналізі програм IoT

Аналіз програми був застосований, або статично або динамічно, до багатьох різних налаштувань, наприклад, до мобільних додатків. При дослідженні п'яти платформ IoT, було виявлено, що вони володіють декількома унікальними характеристиками і проблемами в порівнянні з іншими платформами.

По-перше, у випадку Android, є чітко визначене проміжне представлення (ПП), і аналітик може безпосередньо аналізувати ПП код. Наприклад, популярні структури аналізу, такі як Soot і WALA, які були використані для аналізу вихідного коду програми Android, надають бібліотекам можливість перетворити байт-код Dalvik на Jimple ПП, побудувати графіки викликів, а також виконати аналіз міжпроцедурального потоку даних через доступність графів. Однак платформи IoT програмування

змінюють і кожен використовує свою мову програмування. Таким чином, аналіз повинен враховувати природу прикладних програм IoT і здійснювати аналіз на ньому.

По-друге, IoT додатки контролюють фізичні периферійні пристрої та драйвери. Отже, IoT додатки якісно різні вразливості в результаті обробки фізичних процесів, таких як температура, дим, рух, вологість, витік води, і яскравість. Наприклад, противник може використати здатність пристрою IoT через фізичні канали досягти певного руйнівного ефекту. Для ілюстрації я хочу розглянути додаток, що надає дозвіл на розумне світло, яке підтримує зміну кольору та інтенсивності світла. Додаток може часто перемикати світло і змінювати кольори на різні відтінки, які можуть викликати у користувачів, які мають епілепсію припадок.

По-третє, програми IoT додатки можуть взаємодіяти один з одним, коли вони знаходяться в одному загальному середовищі. Наприклад, два додатки взаємодіють один з одним, коли дія "Вимикання" програми використовується як подія " Вимикання" в іншому додатку. Взаємодія між додатками може призвести до небажаних станів пристроїв, що призводять до порушень безпеки, і піддавати користувачів таким ризикам, як заблоковані двері, при виникненні пожежі.

Вчетверте, тригерні платформи, такі як IFTTT, Zapier і Microsoft Flow все частіше використовується для подолання розриву між фізичними (наприклад, пристроями IoT) і цифровими (наприклад, електронною поштою, соціальними медіа платформами, тощо) процесами. Ці платформи дозволяють користувачам використовувати правила, які з'єднують події та дії пристроїв IoT з подіями та діями цифрових послуг. Наприклад, користувач може використовувати правило, яке публікує твіт, коли вона вмикає світло у кімнаті, відповідно інше правило реєструє присутність користувача до файлу електронних таблиць, коли вхідні двері розблоковано. Це взаємозаплутане середовище розширює взаємодію між пристроями в онлайн-службах. Наприклад, додаток IoT, який підписується на подію "включення"

комутатора, взаємодіє з правилом тригер-дії, який "вмикає" перемикач, коли користувач відмічає на фото на Facebook.

3.4 Аналіз додатків «розумних речей»

Системи аналізу, за винятком FlowFence, використовують додатки «розумних речей» для оцінки.

SmartThings Apps (STA, додатки «розумних речей») розроблені з динамічною, об'єктно-орієнтованою мовою Groovy в середовищі пісочниці. Вона обмежує розробників певною підгрупою мови Groovy для ефективності та безпеки. Наприклад, пісочниця забороняє додаткам створення своїх власних класів і потоків. Сервер хмари створює програмні обгортки для фізичних пристроїв і запускає додатки. STA виконуються в екосистемі розумних речей, аніж у хабі чи хмарі розумних речей. Користувачі можуть встановлювати програми SmartThings з ринку, або з фірмової системи через SmartThings Mobile.

У першому випадку, публікація програми на офіційному ринку вимагає, щоб розробник подав на розгляд вихідний код програми. Офіційні програми з'являються на ринку після завершення тривалого процесу розгляду. В останньому випадку, організації можуть розробити додаток і зробити його доступним за допомогою Web IDE. Ці самостійно опубліковані програми не проходитимуть жодної офіційної перевірки і часто поширюються в офіційному форумі спільноти SmartThings.

Program Analysis of SmartThings Apps. Виконання аналізу програми з вихідного коду вимагає, серед іншого, створення графіка потоку контрольних потоків програми (ICFG). Оскільки Groovy є мовою, яка розміщується на JVM, природно спочатку скомпілювати коду Groovy- в байт-коді Java за допомогою компілятора Groovy, а потім виконати аналіз за допомогою аналітичної бази, такої як Soot. Однак, такий підхід може бути неможливим через інтенсивне використання відображення в байт-коді, сформованому компілятором Groovy. Зокрема, він переводить виклики

прямого методу у виклик шляхом відображення. Система IoT безпосередньо аналізує абстрактне дерево представлення синтаксису джерела Groovy безпосередньо.

3.5 Висновки з розділу 3

В даний час безпека є найбільшим обмежувальним фактором технологічного розвитку в таких областях, як IoT або Cyber-Physical Systems. Ця проблема набуває ще більшого значення, коли оброблювальне навантаження розподіляється між хмарними ресурсами і парадигмою MCC. Найбільш важливими аспектами в цих областях є забезпечення конфіденційності користувачів, конфіденційності даних і забезпечення безпеки додатків, що використовують хмарні ресурси. Ці вимоги відіграють важливу роль, оскільки в наш час IoT характеризується технологіями по наданню інноваційних послуг у різних сферах застосування. Через цю неоднозначність природи розподіленої парадигми IoT дуже складно розробити повністю захищені методи та протоколи для виявлення та запобігання уразливостям і кібер-атакам.

4 СПІЛЬНА РОБОЧА АРХІТЕКТУРА ДЛЯ ДОДАТКІВ, ЗАСНОВАТИХ НА IoT

Нині розробка розширених додатків на основі IoT залишається проблемою. Обробка одночасних потоків даних, обробка даних або виконання складних математичних функцій можуть переповнювати обчислювальні можливості вбудованих систем і мобільних пристроїв.

Один підхід до подолання цього недоліку полягає в розробці розподіленої системи, де сенсорні пристрої є розподіленою частиною для отримання даних і централізованої інфраструктури, яка виконує жорстку обробку. Для цього було розроблено класичну архітектуру клієнт/сервер. В даний час ця централізована інфраструктура зазвичай розгортається в хмарі. Проте, цей зсув вносить кілька нових ризиків, а деякі комунікації між пристроями та централізованою системою можуть стати причиною вузьких місць та затримок. Зокрема, останній недолік має найбільший вплив на мультимедійні дані, наприклад, в додатках, які використовують пристрої отримання відео та зображення. З цієї причини важко впровадити централізовану систему мультимедійного аналізу в хмарі.

Подолати ці вузькі місця і затримки, можливо за допомогою розподіленої архітектури для виконання спільної роботи для середовищ, заснованих на IoT, і використання навантаження програми між доступними пристроями. Ця вдосконалена архітектура враховує різні мережні шари та їхні обчислювальні платформи, від віддалених Cloud-серверів до підключених розумних датчиків і власне «розумних речей». Цей підхід спрямований на оптимізацію використання обчислювальних ресурсів середовища IoT, забезпечуючи при цьому рамки, здатні отримувати дані від датчиків, виконувати складні обчислювальні завдання та запускати розширені програми.

Справа в тому, що концепція та розробка моделей обробки, заснованих на схемах спільної роботи та хмарних обчислень, можуть забезпечити

необхідну обчислювальну потужність для запуску додатків, коли вони працюють на вбудованих пристроях з обмеженою продуктивністю. Додаткове використання інфраструктури хмарних обчислень на вимогу забезпечить гнучкість у виконанні необхідних завдань, а також механізми підтримки технічного обслуговування послуг. Це можливо реалізувати навіть з пристроями та датчиками з низькою обчислювальною здатністю. В цьому допоможе концепція розподіленої архітектури, яка поєднує в собі зондування і обробку на різних рівнях мережі для виконання спільної роботи, заснованої на парадигмі Мобільних хмарних обчислень (MCC).

Цей підхід може бути використаний у різноманітних реальних додатках, що працюють у різних середовищах при різних умовах, де доступний набір обчислювальних систем. Щоб підтвердити дану концепцію, варто розглянути набір експериментів з відкритим набором даних, розділених на кілька підмножин, що дозволяють проводити паралельне тестування. Головною метою цього розщеплення є доведення того, що спільна робота не впливає ані на кінцеву точність, ні на правила чи отримані дані.

4.1 Структурний дизайн для розподілених обчислень

Фреймворк в основному пов'язаний з моделюванням розподіленої системи, протоколами зв'язку, послугами доступу та виявлення і проектуванням методу планування завдань по всій системі. Це може включати в себе специфікацію задіяних пристроїв і вимоги до роботи системи, такі як якість обслуговування (QoS), визначення завдання і обмеження застосування.

Інші підходи розробляють концепцію віртуального сенсора як абстракцію реальних датчиків, відтворюючи їх логічну поведінку і розширюючи їх функціонал програмованими операторами. Використання віртуальних датчиків постійно зростає для переходу з пасивних датчиків до розумних речей а також спрощення створення та налаштування складних

додатків. Ця еволюція дозволяє краще керувати датчиками і полегшує розподіл даних і обчислень серед них.Нові пристрої-посередники для розподілених середовищ спрямовані на поліпшення інтегрованого управління гетерогенними даними, ресурсами та подіями, а також для забезпечення виявлення ресурсів та взаємодії в різних програмах і службах

Нарешті, останні тенденції підштовхують можливості обробки до краю мережі, де аналіз даних та генерування знань виконуються різнорідними процесами. Такий підхід досягає значного скорочення затримки і забезпечує безпеку локальних даних. Що стосується цієї проблеми, то проводяться інтенсивні дослідження з метою адаптації методів планування на основі завдань до вбудованих пристроїв і типів додатків, задіяних у середовищах IoT, для повного використання вузлів і поліпшення продуктивності.

4.2 Комбінація IoT та хмарних обчислень

Хмарні обчислення розглядаються як парадигма резервування ресурсів у розподілених системах. Таким чином, хмарна інфраструктура може централізувати більшу частину обробки інформації. Поєднання IoT і Cloud Computing генерує синергію для обох парадигм і робить об'єкти IoT розумнішими. Інтеграція, що базується на хмарах, повністю забезпечує систему IoT, оскільки центри обробки даних можуть використовувати складні методи машинного навчання та обробки великих масивів даних, для визначення значень вихідних даних. Ця інтеграція під назвою CloudIoT дозволяє надати користувачам потужні додатки в декількох областях реальногосвіту.

Однак система може обробляти стільки даних, допоки система не сповільниться, а затримка буде зростати аж до неприйнятної точки для додатку. Останні підходи намагаються використати обчислювальну потужність пристроїв і аутсорсингу робочого навантаження на хмару тільки тоді, коли це необхідно. Ця тенденція називається Mobile Cloud Computing (MCC). Ця парадигма є перспективним способом поліпшити продуктивність,

а також знизити споживання енергії «річчю», виконуючи деякі фрагменти програми на віддаленому сервері. Це поєднує хмарні обчислення, мобільні обчислювальні системи та бездротові мережі, щоб принести якнайбільше обчислювальних ресурсів користувачам мобільного зв'язку, операторам мереж, а також постачальникам послуг хмарних обчислень. Таким чином, розподіл обробки між пристроями IoT і ресурсами хмарних обчислень здатний збільшити можливості системи IoT і досягти більшої загальної продуктивності при виконанні програми. Існує декілька методів підвищення ефективності та ефективності процесу розвантаження, таких як багатокритеріальний аналіз рішень, стохастичний аналіз або прикладний метод.

Ефективний динамічний розподіл завдань є дуже важливою та важкою темою в середовищі IoT. Іноді потрібна синхронізація між серверним шаром і пристроями або між пристроями. У цьому відношенні push-повідомлення виступають як доставка інформації з програмного забезпечення на обчислювальний пристрій без конкретного запиту з пристрою. Вони надходять з серверного шару, і використовуються для отримання оповіщень обробки в пристрої. Важливою перевагою push-повідомлень у MCC є те, що ця технологія не вимагає, щоб додатки пристрою були відкриті для отримання цих повідомлень. Це дозволяє смартфонам і смарт-одягу приймати і відображати сповіщення про текстові повідомлення, навіть коли їхні екрани заблоковані, а додаток, що отримує push-сповіщення, закрито.

4.3 Дизайн розподілених додатків

Додатки для IoT є розподіленими системами за своєю природою. Таким чином, IoT стає новою інформаційною архітектурою на основі Інтернету, що дозволяє передавати дані та інформацію з реального світу до користувачів і промислових додатків.

Дизайн додатків значною мірою спирається на програмування речей, на рівні операційної системи, що знижує ефективність і надійність програми

IoT. Проте складність конструкції значно знизити при використанні відповідних веб-сервісів.

Нині існує ряд досліджень, що, розглядають хмару як платформу для розгортання розподілених додатків IoT. Основними характеристиками цієї платформи є те, що кожен об'єкт є автономним соціальним агентом; платформа як модель обслуговування (PaaS) повністю експлуатується. Розглядається можливість повторного використання на різних шарах; дані знаходяться під контролем користувачів. Існують також комерційні фреймворки та платформи, призначені для розробки та використання IoT додатків. Важливість цієї платформи дозволяє розробити стандартизовані кінцеві точки та сховища даних, що дозволить забезпечити безпечну взаємодію модульних і розподілених додатків. Ці рішення можуть також підтримувати велику обробку даних і автономне керування під час виконання.

Особливість багаторазового використання дуже важлива, оскільки дозволяє програмістам створювати шаблони речей і їх послуг. Підключені речі можуть використовувати різні типи протоколів і шаблони підключення. Таким чином, шаблони дизайну додають абстрактний шар і полегшують створення надійних і багаторазових рішень.

Розподілені програми зазвичай записують численні набори подій, повідомлень і лог-файлів, роблячи ці платформи відмінним джерелом даних для інструментів для розробки процесів. Тим не менш, розуміння поведінки всіх цих додатків, дуже часто з гетерогенними пристроями, та їх лог-файлів може бути настільки ж складним як і неточним схильним до помилок.

Нарешті, централізовані підходи були, переважно, вибором для забезпечення інформації на основі отриманих даних. Проте через час реагування та мережевих навантажень деякі розподілені програми потребують децентралізованого підходу до інформації, щоб краще відповідати вимогам до навколишнього середовища та додатків. Існують пропозиції щодо переміщення інформації до краю, щоб запропонувати низькорівневу інформацію для програм IoT. Тоді, дацентралізована

мультиагентна система забезпечує способи боротьби з автономністю і неоднорідністю.

4.4 Розподілена обчислювальна архітектура. Загальна схема

Основна мета розподіленої архітектури полягає в тому, щоб використовувати переваги розгорнутої інфраструктури речей і ресурсів хмарних обчислень для зменшення обчислювальних витрат і підвищення загальної продуктивності. Основна ідея полягає в тому, щоб розподілити робоче навантаження програми між сервером і рештою речей за допомогою обчислювальних можливостей, таких як смартфони, носії, планшети, смарт-сенсори та інші вбудовані пристрої. Цей розподіл робочого навантаження між речами дає змогу здійснювати горизонтальне масштабування для зменшення витрат, а не прибігати до віддалених серверів. Таким чином такі види пристроїв виконують більше завдань обробки, ніж серверний шар. Крім того, хмарні обчислення доступні для використання лише в крайньому випадку, якщо це необхідно. У випадку необхідності асинхронної синхронізації між обчислювальними серверами в хмарі та різними пристроями, система розробляє підхід, що ґрунтується на push-повідомленні.

У цьому розділі модель обчислень, для додатків IoT, визначається відповідно до цієї архітектури. Пропозиція, що зосереджена на розподілених додатках, може бути представлені графіком $\mathbb{A} = \{\mathbb{U}, \mathbb{F}\}$, де

- $\mathbb{U}$ <вершина> представляє виконавчі блоки програми. Таким чином, програма IoT може бути розбита на список виконавчих одиниць: $\mathbb{U} = \{u_0, u_2, \ldots, u_{n-1}\}$.
- $\mathbb{F}$ <край> представляє потоки даних, які обмінюються між виконавчими блоками. Потоки даних встановлюють пріоритет між блоками виконання та обсягом обміну даними. $F(i,j) \in \mathbb{F}$, і визначає обсяг даних, що обмінюються між виконавчим блоком і та j.

Виконавчі блоки додатка пов'язані з його здатністю паралельно обробляти дані і завдання. Це дуже важлива особливість для сучасного машинного навчання та підходів до великих масивів даних у програмах IoT, оскільки межові речі можуть значно збільшити продуктивність та витрати системи без необхідності відправляти дані на сервер для централізованої обробки. Наприклад, на рисунку 4.1 показані три випадки застосування ($\mathbb{A}_i$), модельовані відповідно до цього принципу, де функція фрагментації даних генерує більше одиниць виконання і відкриває більше можливостей обробки серед речей.

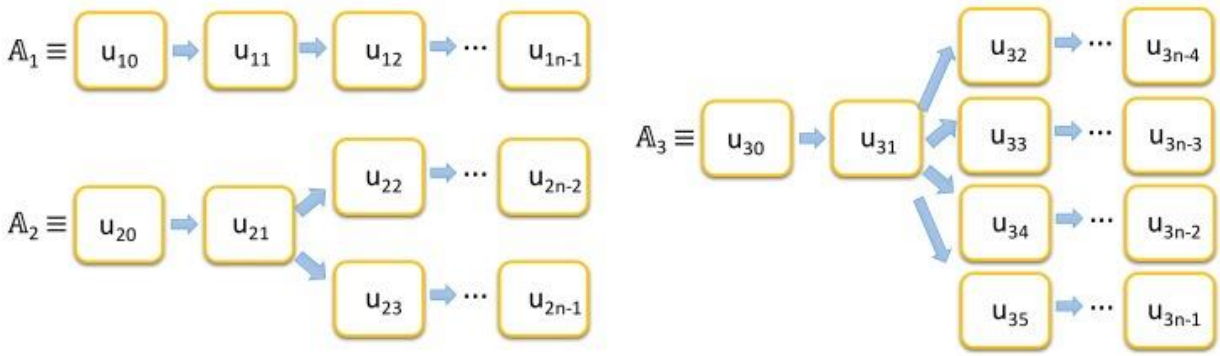

Рисунок 4.1 Діаграма різних можливостей потоку в запропонованій розподіленій системі

Відповідно до цієї моделі, пристрої $\mathbb{D}$, що беруть участь у програмах IoT, визначаються наступним чином:

Нехай S - набір датчиків. Ці пристрої самі по собі не мають обчислювальних можливостей. Їх робота полягає у зондуванні та передачі даних до інших пристроїв або хмари.

Нехай P - набір доступних обчислювальних платформ. Цей набір включає в себе речі, які мають можливості обробки. Пристрої P-набору також можуть збирати дані і обробляти їх.

Нехай C - набір ресурсів хмарних обчислень. У цьому наборі знаходяться віддалені сервери, тобто бробка навантаження знаходиться зовні.

Тобто: $\mathbb{D} = \{S\} \cup \{P\} \cup \{C\}$

Елементи цих наборів взаємопов'язані між собою, створюючи мережу зв'язку IoT. Модель архітектури розподіляє виконавчі блоки $\{\mathbb{U}\}$ через доступні пристрої $\mathbb{D}$ відповідно до їх конкретних можливостей і обмежень додатків. Рисунок 4.2 ілюструє цю ідею.

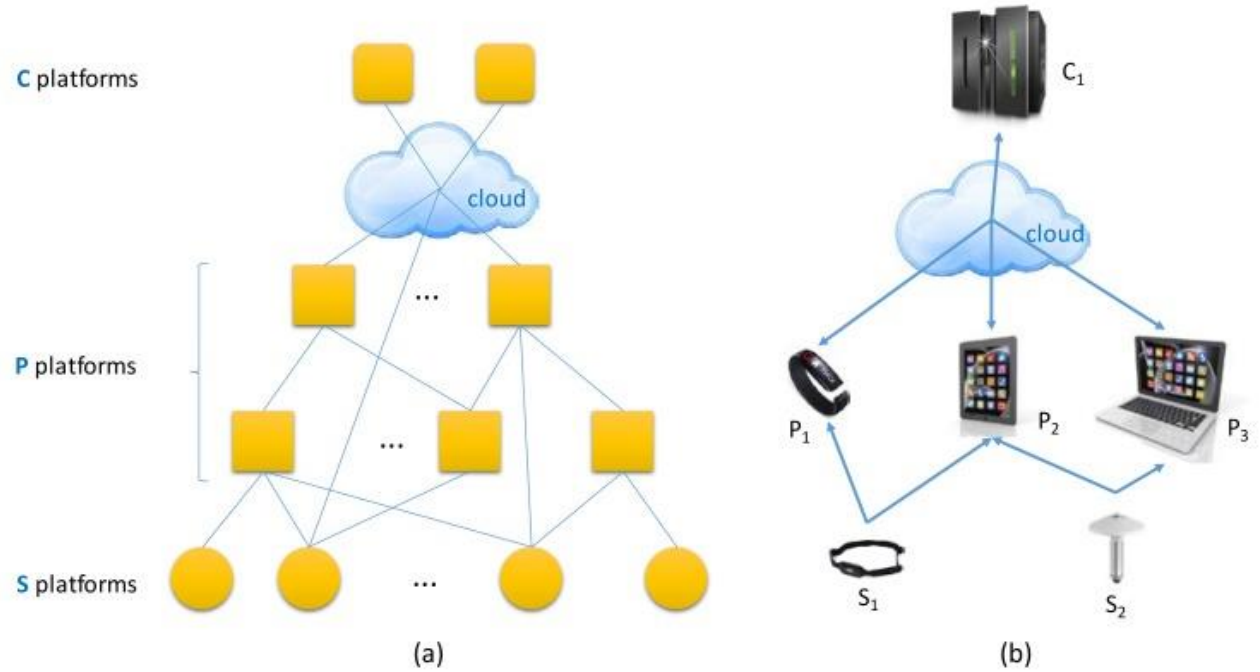

Рисунок 4.2 Мережа зв'язку IoT. (a) загальна схема; (b) Приклад

Поведінку середовищ IoT можна описати як динамічну, оскільки нові речі можуть з'являтися і зникати. Таким чином, необхідна служба виявлення речей для реєстрації нового пристрою та скасування реєстрації, коли він відсутній. Ця служба будує набір $\mathbb{D}$ і, отже, вона відіграє значну роль у розробці додатків IoT, оскільки вони можуть дозволити клієнтам і програмам отримувати доступ до доступних ресурсів і даних, що надаються речами. Ця служба виявлення може бути централізованою та/або децентралізованою. У цій роботі передбачається, що належна служба виявлення виконується для підтримки оновлення набору $\mathbb{D}$.

Для ефективного розподілу ресурсів і вдоволення вимог до застосування, необхідно правильно охарактеризувати використання ресурсів. З цієї причини запропоновано вектор $\mathbb{V}$, який визначається як векторний домен відповідних ознак, що моделюють поведінку обчислювального

навантаження на кожному пристрої. Для додатків, на яких ця робота фокусується, $\mathbb{V}$ може бути визначений як область векторів з двома компонентами в діапазоні [0, 1], з наступною семантикою:

$$\mathbb{V} = \text{Час\_відгуку} \times \text{Швидкість\_передачі}. \quad (1)$$

«Час_відгуку» комп'ютера визначається пропорційно часу виконання, що необхідний вибраному блоку виконання тестів у комп'ютері (0 і 1 відображаються в ситуаціях, коли процесор практично бездіяльний і неприйнятно зайнятий, відповідно). Задача тестування не обмінюється даними через мережу, тому не враховує затримки мережі, а лише локальні ресурси, які в основному і є процесором.

«Швидкість_передачі» визначається як частина доступної смуги пропускання, яка використовується для обробки блоку виконання. Ця частина включає в себе вхідні та вихідні дані блоку виконання. Метод кількісного визначення «Швидкість_передачі» буде враховувати характеристики взаємозв’язаної мережі. Поточна швидкість передачі даних може бути відома для кожного пристрою шляхом моніторингу мережі. Іншими методами може бути використання даних історії у відповідності з умовами експлуатації в кожному конкретному випадку і ситуації, або використання усереднених даних, отриманих при регулярних перевірках. У багатьох випадках набір пристроїв P і S з'єднаний з бездротовою локальною мережею (WLAN), і, отже, затримка зменшується і швидкість передачі мережі дуже часто вище, ніж з'єднання через Інтернет з хмарними серверами.

Інші компоненти можуть бути визначені для того, щоб визначити різні значення продуктивності. Наприклад, «Споживання_енергії» речей, «Ціна_використання» ресурсів, «Ризики_безпеки» обміну даними і т.д.

Кожен пристрій середовища IoT, $d_i \in \mathbb{D}$, має рівень продуктивності, який може бути охарактеризований функцією “ *Perf* ”, яка визначається наступним виразом:

$$Perf: \mathbb{D} \longrightarrow \mathbb{V}. \quad (2)$$

Значення функції *Perf* може бути відоме до виконання програми чи бути динамічною протягом всього часу. Наприклад, смартфон буде

показувати високі значення для «Час_відгуку», якщо він зайнятий певними процесами користувача. У цьому випадку функція продуктивності повинна періодично оновлюватися для прийняття правильних рішень. Інші пристрої можуть мати стабільний рівень продуктивності завдяки їх спеціалізації або здатності абсорбації нових завдань. Статичні дані можуть бути збережені в таблиці пошуку (LUT) для швидкої оцінки функції.

Як тільки обчислювальне навантаження характеризується нормалізованими і відносними значеннями, кожен пристрій середовища IoT показує свою здатність виконувати виконавчі блоки (u) в 𝕌 для застосування однорідним чином.

Наступний підрозділ описує, як виконувати обробку блоків серед доступних обчислюваних «речей» і віддалених серверів.

4.5 Приклад розподіленої обчислювальної архітектури

У цьому розділі я представлю простий тематичний приклад, де пропонується розподілений метод для обробки програми, де відбуваються зондування та складні обчислення.

Додаток, що використовується як приклад, складається з методу аналізу ступеня уваги учнів у класі. Ця інформація контролюється лектором, коли він/вона дає учням завдання. Таким чином, вчитель може знати, хто з студентів і коли саме втомився або втратив увагу, колективно і індивідуально. Хоча загалом це ілюстративний приклад переваг перенесення навантаження на обробку та сприяння спільній роботі між речами, дана проблема в значній мірі вивчається в області безпеки руху, оскільки ступінь уваги водія є ключовою особливістю запобігання аварій на автомобілі. Детектори втоми водія включають в себе як преміальні особливості високоякісних автомобілів. Ці детектори зазвичай базуються на аналізі характеристик обличчя водія, таких як очі і позіхання. З цієї причини типово встановлений датчик є цифровою камерою.

Звівши цю проблему до питання контролю студентської уваги, на рисунку 4.3 показано просту схему програми.

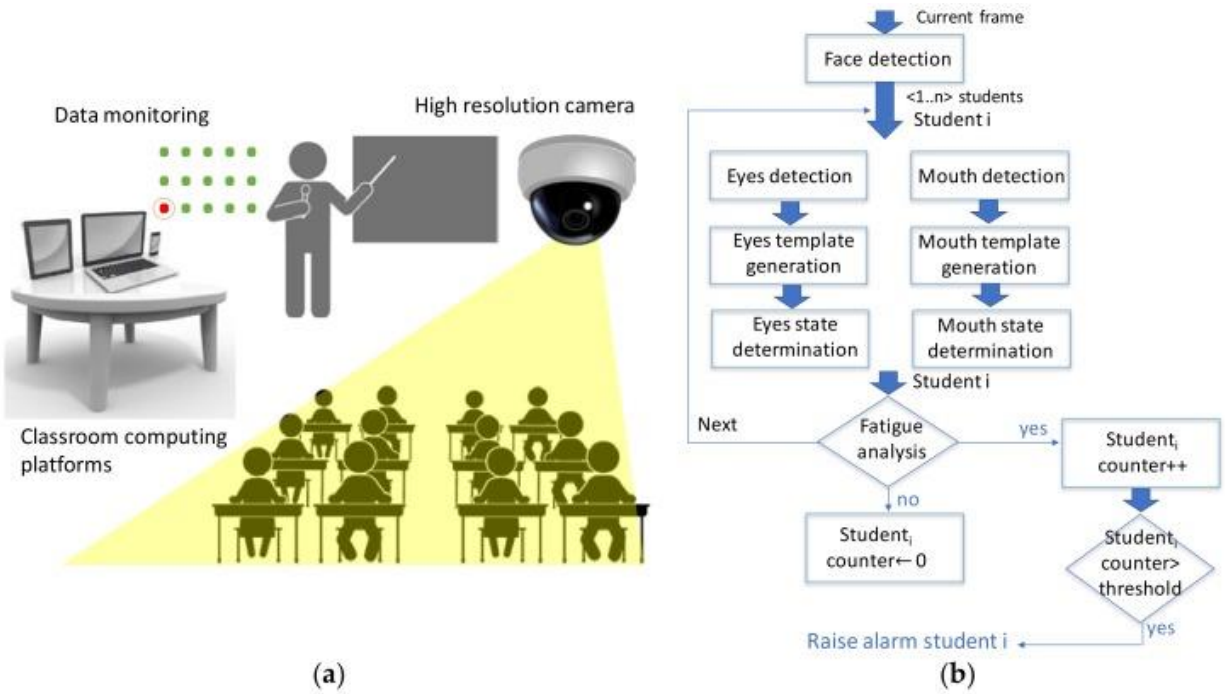

Рисунок 4.3 Схема застосування: (a) Контекст застосування додатку в аудиторії; (b) Етапи застосування додатку

Що стосується конфіденційності цієї заявки, існують дві проблеми, які повинні бути вирішені: по-перше на це повинна бути надана згода, оскільки дані користувача, а саме обличчя, контролюються і обробляються; по-друге, обличчя студентів повинні оброблятися анонімно. Тобто система може зробити висновок, коли ступінь уваги конкретного студента низька і вказати на його/її положення в класі, як показано на рисунку 4a. (червона крапка), але вона не може визначити, хто цей студент, і пов'язати це з будь-якими академічними даними.

Для цього додатка список виконавчих блоків $\mathbb{U}$ має два виміри: він складається зі списку етапів застосування, зображених на рисунку 4b, і для кожного студента в класі.

З іншого боку, потоки даних $\mathbb{F}$, циркулюють між виконавчими блоками, і є фреймами, вікнами фреймами (коли вказано обличчя) і шаблонами даних. Крім того, цей додаток дозволяє проводити високу

паралельну обробку, оскільки очі і рот можуть оброблятися паралельно, і кожен студент також може бути проаналізований паралельно. Цей високий паралелізм полегшує співпрацю між різними обчислювальними платформами.

Доступні обчислювальні платформи для цього додатка можуть бути розгорнуті на декількох шарах, наприклад, кожна класна кімната обладнана портативним ПК, і, зрештою, вчитель може мати планшетний ПК і смартфон, які можуть бути залучені в розрахунки. Крім того, школа могла б мати певну робочу станцію для обробки всіх аудиторій вчасно, і, у випадку накладних витрат, обчислювальні потужності тимчасово винаймаються з зовнішнього хмарного сервера. Застосовується камера з високою роздільною здатністю для створення кадрів, де можна одночасно шукати та виявляти декілька облич поміж всіх учнів.

Наступний рисунок зображує схему всієї системи.

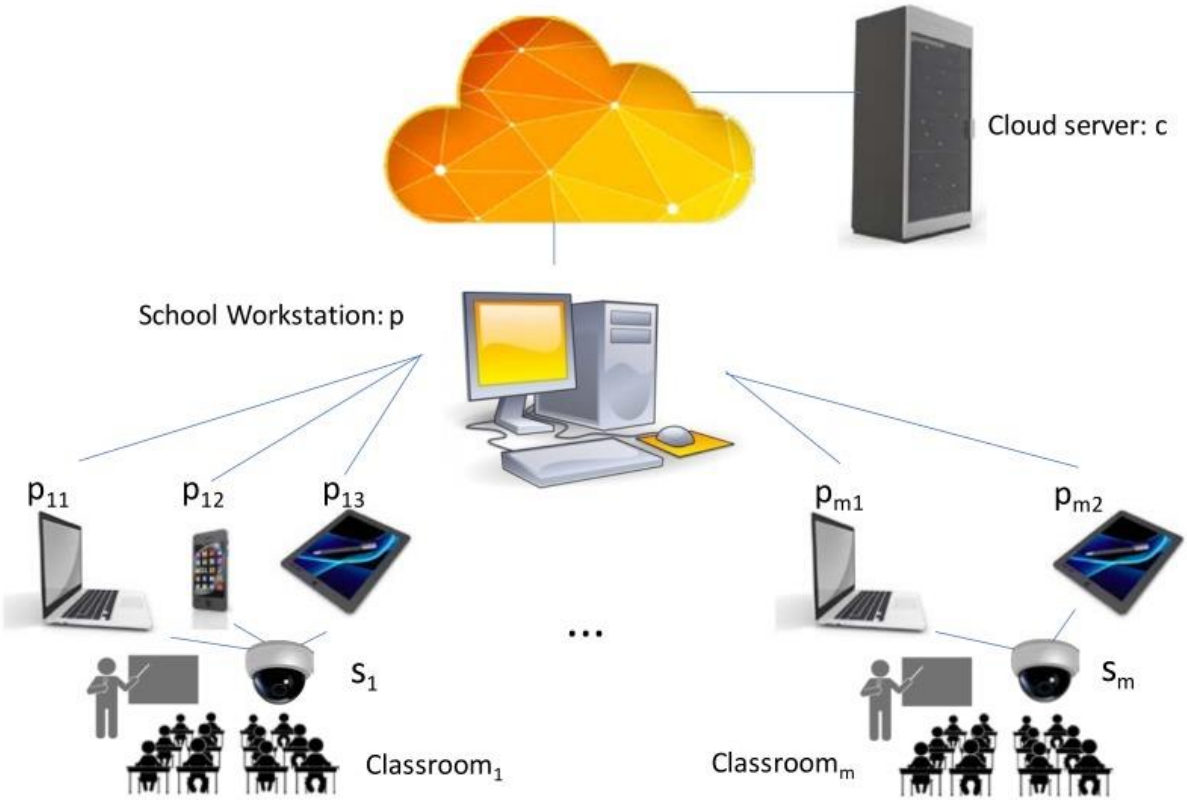

Рисунок 4.4 Прикладне середовище IoT

Схема, зображена на рисунку 4.4, показує можливий розподіл пристроїв, де кожен клас має певний набір доступних пристроїв. Наприклад, Класс$_1$ має три локальні обчислювальні платформи, а Класс$_m$ має лише два.

Дане середовище IoT складається з таких пристроїв:

Набір датчиків {S} = {s_i: камера з високою роздільною здатністю у класній кімнаті i}

- Обчислювальні платформи {P} = {p_{ij}: обчислювальна платформа і класу j, p: шкільна робоча станція}
- Хмарний сервер {C} = {c_0}

Пристрої {S} і {P} знаходяться в мережі зв'язку IoT, підключеної до шкільної мережі WLAN. Доступ до хмарного сервера здійснюється через Інтернет.

Приймемо час обробки для пошуку обличчя у всьому зображенні для одного користувача приблизно 0,5 с для стандартної робочої станції. Передбачається той же час для аналізу рота в фреймі. Таблиця 4.1 показує порівняльний розрахунковий час обчислення кадру для кожного типу обчислювальної платформи відповідно до стандартної апаратної конфігурації кожного типу пристрою. Передбачається класна кімната з 25 учнями та школа з 12 класами.

Таблиця 4.1 Порівняльний розрахунковий час обчислення фрейму для кожного типу обчислювальної платформи відповідно до стандартної апаратної конфігурації кожного типу пристрою

Платформа обчислення	Час обробки фрейму	Пороговий сигнал=5
Комп'ютер у класній кімнаті[1]	25 с	~2 хв.
Планшет у класній кімнаті[1]	50 с	~4 хв.
Смартфон у класній кімнаті[1]	50 с	~4 хв.

Продовження таблиці 4.1

Платформа обчислення	Час обробки фрейму	Пороговий сигнал=5
Шкільна робоча станція[2]	5 хв.	25 хв.
Ресурси класної кімнати[1]	13 с	~1 хв.
Хмарний сервер[3]	25 с+5 с	2,5 хв.

При чому: 1 – Загальний час для 25 студентів. 2 – Загальний час для 12 аудиторій по 25 студентів. 3– Загальний час для 12 класів з 25 студентами плюс затримка зв'язку.

Час зв'язку між платформами в мережі WLAN тут не враховується. Для обробки хмарних обчислень необхідно включити додаткову затримку для передачі фреймів (або вікон фреймів). Інший недолік виникає через проблеми безпеки при використанні ресурсів Cloud. Для того, щоб задовольнити вимоги GDPR щодо обробки даних користувачів, хмарна інфраструктура повинна бути розгорнута в безпечних регіонах, які забезпечують достатній рівень захисту даних. Таким чином, щоб зберегти обробку завантаження кадрів всередині WLAN, виникає ризик витоку даних при обробці поведінки студента.

Час затримки оцінювання, показаний у таблиці 4.1 дає більш точну картину ситуації. Школа може розгортати більше ресурсного обладнання для прискорення розрахунків. Проте замість цього, запропонована архітектура IoT може бути використана, щоб максимально використати існуючі ресурси класу. Спільне використання трьох обчислювальних платформ у класі спільно може значно скоротити час застосування. Подібна, спільна, робота підтримує продуктивність при застосуванні аналізу втомлюваності приблизно на рівні 1 хв., використовуючи лише місцеві ресурси, тобто, мобільний ПК, настільний ПК і смартфон, обробляючи всіх студентів паралельно. Шкільна робоча станція може бути використана для надання допомоги у класах, централізації всіх даних і виведення статистичної інформації про інтереси кожного уроку.

Виконавчі блоки, задіяні в цьому додатку, будуть відповідати за обчислення частини набору даних, наприклад, частину фрейму, де розміщений кожен студент. На рисунку 4.5 показана схема цієї спільної роботи.

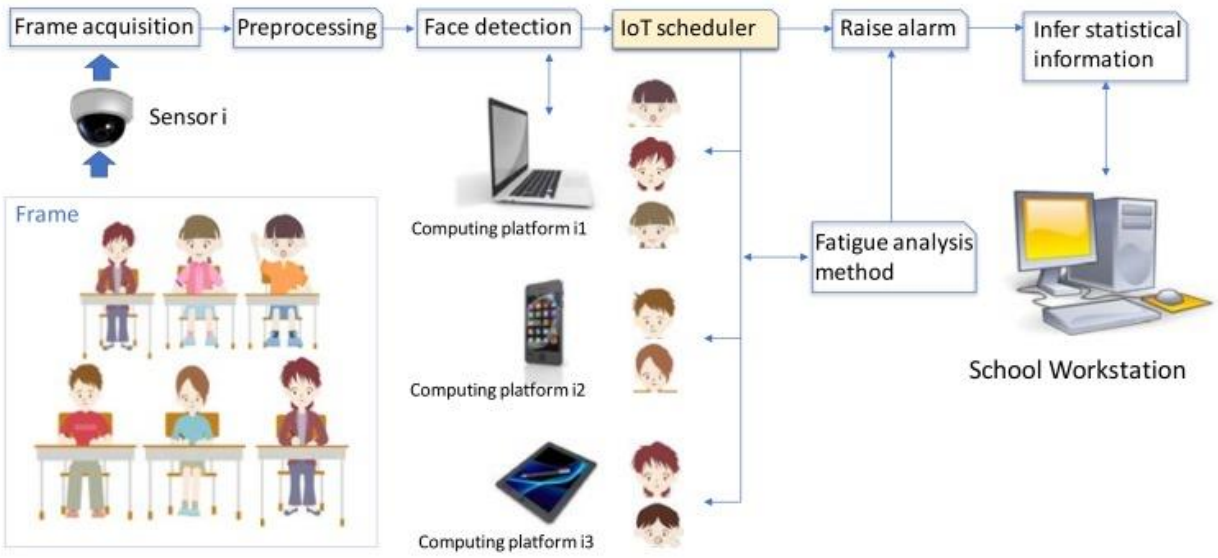

Рисунок 4.5. Приклад спільної роботи

Планувальник архітектури IoT може бути обчислений на класному ПК. Виходячи з попередніх можливостей для належної обробки цього IoT додатку, можлива наступною реалізація компонентів планувальника:

- Пристрої-кандидати: викладач може мати планшет та/або смартфон, які можуть бути частиною програми. Таким чином, цей модуль визначає список платформ, що знаходяться на класі (класний мобільний ПК, настільний ПК, смартфон), а також шкільну робочу станцію та хмарний сервер.
- Модуль оцінки продуктивності: Продуктивність задіяних пристроїв у даному випадку відома, і додаткові пристрої не можуть приєднатися до цієї системи в кожному класі. Тому ця функція просто визначає доступну ємність кожного пристрою відповідно до їх поточного навантаження.
- Модуль Perf-LUT: смартфон і планшет викладача мають оцінюються в автономному режимі для оновлення цього модуля. На високому рівні,

наприклад, вміст цього модуля показаний в Таблиці 4.1. Більш точно, LUT зберігає час обчислення кожного етапу (блоку виконання) для аналізу ступенів уваги студентів. Паралельна функція цього додатка дозволяє обробляти частину фрейму кожного студента як блок виконання.

- Функція продуктивності (Perf): Поведінка цієї функції може бути визначена заздалегідь, щоб проводити обробку даних якомога ближче до місця їх отримання. Таким чином, пристрої сортуються по близькості і результат витягується з модуля оцінки продуктивності. У загальних рисах, класна кімната з мобільним ПК повинна бути обрана в першу чергу, далі класна кімната з планшетним ПК, і нарешті класна кімната з смартфоном. Коли ці пристрої зайняті, робоче навантаження може бути передане на централізовану робочу станцію та хмарний сервер.

Цей приклад показує можливості спільного розгортання та обчислення додатків для запропонованої архітектури. Як показано, архітектура використовує обчислювальні можливості доступних пристроїв IoT для прискорення обчислень і досягнення кращої продуктивності, ніж централізованим способом. Крім того, ця альтернатива не може бути доступною через перевантаження мережі або проблеми конфіденційності.

4.6 Висновки з розділу 4

Загальною проблемою є те, що обчислювальні вимоги для моніторингу та вдосконаленого аналізу даних, отриманих за допомогою IoT, зазвичай перевищують можливості датчиків і навіть мобільних комп'ютерів. Саме тому розподілена архітектура, поєднує в собі зондування і обробку даних на різних рівнях мережі для обміну обчислювальним навантаженням серед доступних пристроїв. Середовище IoT, що складається з носіїв та інших біосенсорів, може скористатися цією архітектурою, дозволяючи розширену та спільну обробку даних з додатківпри обмеженнях реального часу.

Окрім узагальнення відповідних та репрезентативних внесків, а також передумови розподілених обчислень IoT, дане дослідження, є подальшим кроком для розробки схем співпраці прикладних програм IoT. Головною перевагою даної системи є те, що вона дозволяє здійснювати моніторинг і аналіз у реальному часі всіх отриманих даних мережевими пристроями. Крім того, вона забезпечує гнучкість у виконанні програми за допомогою ресурсів з хмарних обчислювальних служб, а також з інших доступних комп'ютерів, які рекомендуються для зниження собівартості або кращої доступності. Ключовими новинками є формалізація програми та її декомпозиція у виконавчих одиницях, а також новий модуль планувальника. Це основний компонент архітектури для розподілу виконавчих блоків і їх передачі відповідно до обчислювальних можливостей кожного пристрою.

Ще однією цікавою проблемою, яку слід розвивати, є стійкість системи. Запропонована структура може підтримувати резервацію, яка буде активована, коли центральний контроль недоступний або недосяжний. Ця ідея може бути розширена шляхом додавання механізму виявлення сервісу, що дозволить комп'ютерам автономно приймати оптимальні рішення на основі локальної інформації, що надходить від сусідніх пристроїв.

ВИСНОВКИ

Після ознайомлення з репрезентативними пропозиціями в цій області можна визначити деякі висновки:

- Кількість пов'язаних речей значно зростає. Це збільшує можливості розробки сучасних додатків, які використовують переваги їх всюдисущого зондування та обчислювальних можливостей.
- Обчислювальні ресурси всієї мережі можуть бути використані для підвищення продуктивності додатків на основі IoT шляхом спільного використання навантаження на обробку між доступними платформами та способу більш інтенсивного використання розгорнутої інфраструктури.
- Незважаючи на прогрес, досягнутий останніми дослідженнями, належний розподіл навантаження на додатки залишається проблемою,існує певна відсутність формалізації та узгоджених механізмів реалізації реальних спільних додатків для середовищ IoT.

У цьому контексті представлене дослідження є кроком вперед у розробці спільних схем додатків IoT та Cloud шляхом спільного використання робочого навантаження програми між пристроями IoT середовища. Запропоновано новий модуль формалізації та планування програми для обробки робочої співпраці між різнорідними речами та іншими мережевими ресурсами. Компоненти планувальника деталізовані і описаний практичний випадок використання. Інтеграція хмарних обчислень і IoT свідчить про наступний великий стрибок у світі Інтернету. Нові додатки, що переповнені цією комбінацією, відомої як IoTCloud, відкривають нові шляхи для бізнесу, а також для досліджень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Collaborative Working Architecture for IoT-Based Applications / Higinio Mora, María Teresa Signes-Pont, David Gil, Magnus Johnsson, 2018. [Електронний ресурс]. – Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6022002/

2. How Fog Computing Powers AI, IoT, and 5G, Madhavan Sridharan, April 9, 2019. [Електронний ресурс]. – Режим доступу: https://www.datastax.com/2019/04/how-fog-computing-powers-ai-iot-and-5g

3. How the introduction of IoT applications is transforming various industries and markets, 2019. [Електронний ресурс]. – Режим доступу: https://channels.theinnovationenterprise.com/articles/the-internet-of-things-iot-applications-that-are-changing-the-future

4. Internet of Things: Architectures, Protocols, and Applications / Pallavi Sethi, Smruti R. Sarangi // Journal of Electrical and Computer Engineering olume – 2017. [Електронний ресурс]. – Режим доступу: https://www.hindawi.com/journals/jece/2017/9324035/

5. Importance of Cloud Computing for Large Scale IoT Solutions, 2018. [Електронний ресурс]. – Режим доступу: https://www.einfochips.com/blog/importance-of-cloud-computing-for-large-scale-iot-solutions/

6. OneM2M -A Common Service Layer for IoTBasic principles and architecture overview, 2018. – [Електронний ресурс]. – Режим доступу: https://portail-qualite.public.lu/dam-assets/publications/normalisation/2018/workshop-etsi/4-xavier-piednoir-onem2m-workshop-ilnas-etsi.pdf

7. Program Analysis of Commodity IoT Applications forSecurity and Privacy: Challenges and Opportunities, 2018. – [Електронний ресурс]. – Режим доступу: http://www.yurradnik.com.ua/stride/ur/index.php?m=archive&y=2005&mag=4&art=89

8. Role of Cloud Backend in IoT and Basics of IoT Cloud Applications, 2018. . [Електронний ресурс]. – Режим доступу: https://www.embitel.com/blog/embedded-blog/role-of-cloud-backend-in-iot-and-basics-of-iot-cloud-applications

9. Program Analysis of Commodity IoT Applications forSecurity and Privacy: Challenges and Opportunities, 2018. – [Електронний ресурс]. – Режим доступу: http://www.yurradnik.com.ua/stride/ur/index.php?m=archive&y=2005&mag=4&art=89

10.What is the Cloud? How Does it Fit into the Internet of Things (IoT)?, 2019. [Електронний ресурс]. – Режим доступу: https://www.iotforall.com/what-is-the-cloud/